International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 4, Issue 2, Apr 2014, 223-232

INTRUSION PROTECTION AGAINST SQL INJECTION ATTACK AND CROSS SCRIPTING ATTACK USING A HYBRID METHOD

ASHWINI LOKARE & S. B. WAYKAR

Sinhgad Institute of Technology, Lonavala, Pune, Maharashtra, India

ABSTRACT

Keeping in mind the increasing volume of real time transactions on the internet, security in Web Applications is vital to protect the value and usability of assets. The level of security has neither grown as fast as the Internet Applications nor evolved as fast as the attacks and intrusions, exposing various vulnerabilities inherent in the Internet bases services of the era.

These Application-level exposures have been exploited with serious consequences including shipping goods for no charge, thefts and leaks of confidential data.SQL Injection and Cross-Site Scripting are the two most common attacks that exploit the vulnerabilities in Web Applications. We have proposed a combination of CIDT (Code Injection Detection Tool) & Reverse Proxy to protect against SQL Injection and Malicious Code Injection Attacks.

We propose to achieve this by utilizing a second server of reverse proxy to increase the efficiency of the web server. This technique has the advantage that it can be used as an add-on tool for most web applications without the need to make changes in the initial coding.

KEYWORDS: SQL Injection, Malicious Code Injection Attacks, Cross-Site Scripting

INTRODUCTION

Code Injection is a type of attack in a web application, in which the attackers inject or provide some malicious code in the input data field to gain unauthorized and unlimited access, or to steal credentials from the users account. The injected malicious code executes as a part of the application. This results in either damage to the database, or an undesirable operation on the internet. Attacks can be performed within software, web application etc, which is vulnerable to such type of injection attacks. Vulnerability is a kind of lacuna or weakness in the application which can be easily exploited by attackers to gain unintended access to the data [2]. Some common code injection attacks are HTTP Request Splitting Attacks, SQL Injection Attacks, HTML Injection Attacks, Cross-Site Scripting, Spoofing, DNS Poisoning etc

PROPOSED METHODOLOGY

INTRODUCTION

Integrated technique is proposed in here which deals with both the Code Injection attacks, caused via Vulnerable Web Applications. The proposed system has two modules; the Script detector and Query detector which is placed in proxy server. HTTP request coming from the client side instead of going to the web server is transferred to proxy server within which the request is feed to CIDT which having both modules one by one. And when, any malicious content is found in the request by either of the module, the request is considered as invalid and its executions prevented on the web server.

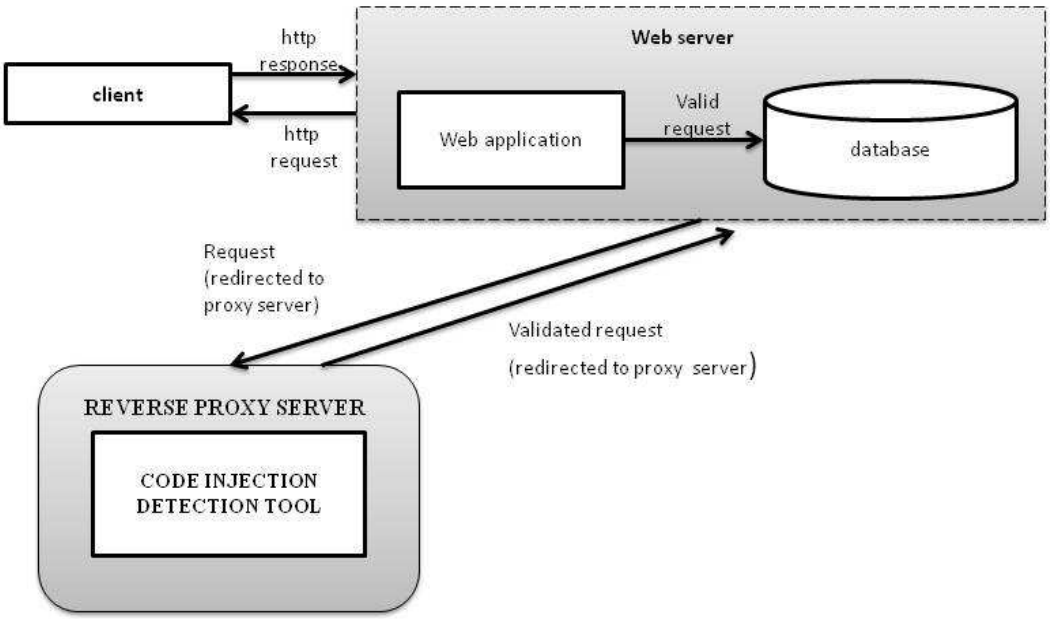

Figure 1: System Architecture

In a client server model, a reverse proxy server is placed, in between the client and the server. The presence of the proxy server is not known to the user. The sanitizing application is placed in the Reverse proxy server A reverse proxy is used to sanitize the request from the user. When the request becomes high, more reverse proxy's can be used to handle the request. This enables the system to maintain a low response time, even at high load.

The general work of the system is as follows:

- The client sends the request to the server.
- The request is redirected to the reverse proxy.
- The CIDT application in the proxy server extracts the URL from the HTTP and the user data from the SQL statement.
  - The URL is send to the signature check
  - The user data (Using prototype query model) is encrypted using the MD5 hash.
- The application sends the validated URL and hashed user data to the web application in the server.
- The filter in the server denies the request if the application had marked the URL request malicious.
- If the URL is found to be benign, then the hashed value is send to the database of the web application.
- If the hashed user data matches the stored hash value in the database, then the data is retrieved and the user gains access to the account.
- Else the user is denied access. Figure gives the flowchart of the system.

Flow Chart of Propose Technique

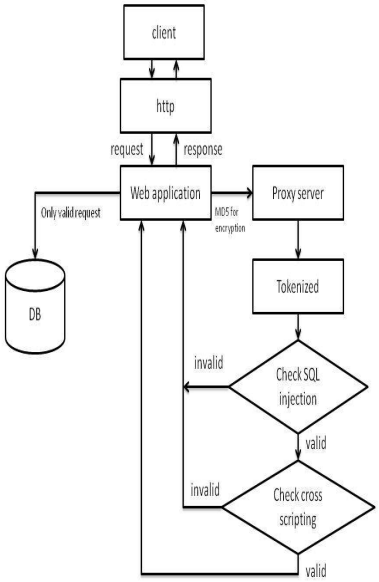

Figure 2: Flow Chart

The block diagram of proposed system is shown in figure 2

- CIDT functions like a proxy between user request and web server. The HTTP request having a session id is forwarded to the proxy agent (CIDT), which authenticates the request by sending it to the Query detector and Script Detector.
- First, Script detector validates the request and if any invalid character is found in the input query it is rejected and not forwarded to the next module. Only request which are reported as valid by Query detector are forwarded to the next module.
- Script detector filters the request for invalid tags and encodes it before forwarding to the server.
- Functionality of both the modules is independent in a sense that the valid request goes to both the modules before getting executed on the web server.

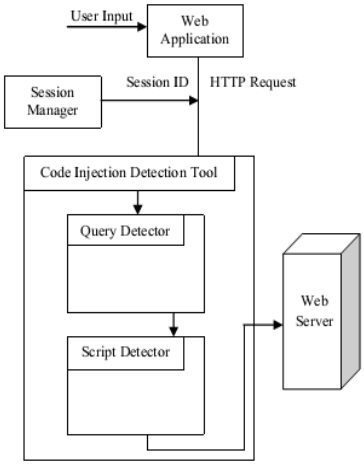

Figure 3: Block Diagram of Code Injection Detector

Query Detector

A Query Detector is a simple tool which is used to test the precision of SQL Queries, and detecting malicious Request from user at the web server. It takes request coming from any user and validates the request before forwarding it to the web server for further execution and processing.

Session Manager

When HTTP request goes to the web server a Session object for that user is initialized [2], which assign a Session variable or Token for that particular connection. This session remains in its active state until the connection remains active. As soon as the connection is terminated the session terminates

Input Valuator

Input Valuator is a key section of Query detector. It works as a Proxy between Client and the web server and any request going on the web server is first validated at the Input Valuator. It has an attack vector repository consisting of some special characters (e.g. ' - ;) which are often used in writing malicious code for SQL Injection attack. It does the functionality of matching user supplied data in HTTP request with the text file stored in attack repository. When user supplied text contain any special symbols which are present in the repository, it is treated as invalid request by the Input Valuator. Execution of that request on the web server is prevented. If no pattern is matched then that request is treated as valid and is forwarded to the next module for filtering the script tags.

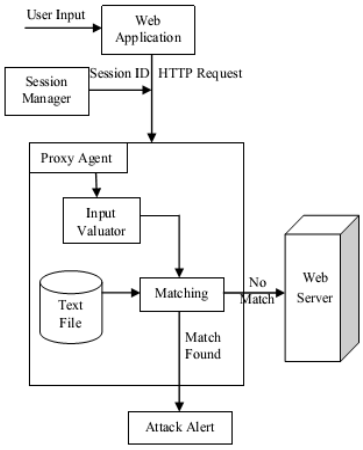

Figure 4: Block Diagram of Query Detector

Script Detector

Script detector is used to detect the malicious script embedded in the web application. It sanitizes HTML input before executing on the web server. This sanitization process removes all the invalid and unwanted tags from the user input and then encodes the remaining input into simple text thus preventing the execution of any malicious script. The block diagram of Script Detector shown in figure 3 has different blocks which prevent the Cross-Site Scripting attack.

HTML Sanitizer

HTML Sanitizer removes unsafe tags and attributes from HTML code. . It takes a string with HTML code and strips all the tags that do not make part of a list of safe tags. The list of safe tags is defined according to the white list tags list given by Open Web Application Security Project (OWASP) [3].

There are some functions to disallow unsafe or forbidden tags like script, style, object, embed, etc. It can also remove unsafe tag attributes, such as those that define JavaScript code to handle events. The links href attributes also gets special treatment to remove URLs that trigger JavaScript code execution and line breaks. The list of all the allowed tags and forbidden tags is given in Table 2. The sanitization process starts with breaking the HTML string in tokens; this functionality is handled by HTML tokenizers.

Tokenizer

Tokenize divides the HTML text within user input into tokens. A token is a single atomic unit of supplied text. In proposed method a token is be one of the following: tag start (), comment (), tag content (“text”), a tag closing (). As a result of this a list of tokens will be created, and then each and every token in this list is matched with the white list tags and forbidden tags shown in Table 2. And then the HTML Sanitizer forward’s the user request to HTML Encoder

HTML Encoder

HTML encoder performs the character escaping. It uses the Html Encode Method of ASP.NET to encode the user input. The Html Encode method applies HTML encoding to a string to prevent a special character to be interpreted as an HTML tag. This method is useful for displaying text that contain "special" HTML characters such as quotes, angular brackets and other characters by the HTML language. Table 1 Shows a list of some of these special characters and their equivalent encoded value, which is used by the HTML Encoder to encode the input.

Script Pattern

This contains all the tags and patterns that are used to match with the tokens which are formed by the tokenizer. It contains list of all the forbidden tags, allowed tags, tag starting pattern, tag closing pattern, comment patterns, style pattern, URL pattern etc. The list of all patterns used by this module is shown in Table 2.

Pattern Matcher

The functionality of this module is just to take the input from the list of tokens and match them with the Script Patterns. All the rejected tags are stored in the invalid tags list and all the accepted tags are forwarded to the HTML Encoder for encoding

ALGORITHMIC MODEL

MD5 HASH KEY ALGORITHM

MD5 Algorithm Description

MD5 algorithm takes input message of arbitrary length and generates 128-bit long output hash. MD5 hash algorithm consist of 5 steps

ALGORITHM 1: MD5 HASH KEY

- **Step 1:** Append Padding Bits
- **Step 2**: Append Length
- **Step 3**: Initialize MD Buffer
- **Step 4**: Process Message in 16-Word Blocks

- **Step 5**: Output

QUERY DETECTOR ALGORITHM

ALGORITHM 2: QUERY DETECTOR

\begin {SQL_Detect}

Step 1: Accept u_name, u_pass in text from users.

Step 2: Start the Session for current u_name.

Step 3: Forward u_name

Step 4: Set attack ← False;

Step 5: Repeat <for each line of input>

Until {(line equal to Test.txt) and not equal to Null}

\End While

Step 6: Set line ←String Pattern;

Step 7: If {u_name.contains(line)}

Set attack ← true;

\End If

Step 8: If {attack equals to true}

Set Valid ←false;

Else

Set Valid ←true;

\End If

Step 9: If {Valid is equal to false}

Discard U_name from entering into the database.

Else

Allow Connection to database.

\End

User's request through a web application is forwarded to the Query Detector. Algorithm 2 then matches the content of user request with the text file for any special character. If any special character gets matched, the request is said to an invalid request and its execution is stopped. Otherwise it is allowed to be executed

SCRIPT DETECTOR ALGORITHM

ALGORITHM 3: SCRIPT DETECTOR

Step 1: Take user input in the form of any HTML text having scripts, tags, links, or urls.

Step 2: Tokenize the input code.

Step 3: Store all the tokens in a list.

Step 4: Having the list of token, check for every single token whether it is acceptable or not.

Repeat {for every token check it with a regular expressions}

a) If token is a comment discard it.

b) If {token is a start tag}

Extract the tags and all its attributes

If {Forbidden Tag}

Remove the tag.

\End if

If {Allowed Tag} then do

- Extract every attribute of the tag.
- Check the “href” and “src” for admitted tags.(a, img, embed)
- Check the “style” attribute and discard it.
- Remove every “on…..” attribute (onclick, onmouseover…)
- Encode attribute value for unknown ones.
- Push the tag on the stack of open tags.

Else

The tag is unknown and will be removed.

\End If

If {token is a end tag} then do

Extract the tag

Check whether the corresponding tag is already open.

Else

It is not a tag encode it.

\End If

\End While

Algorithm 3 describes the process of sanitization.

Sanitization is a process of filtering html content present in the input request. The function of sanitizer is to tokenize the user request and collects the list of tokens. Each token is matched with the script pattern using regular expressions. Unwanted or invalid tokens are removed from the user request and then the system encodes it before forwarding to the web server.

ADVANTAGES OF INTEGRATED SYSTEM

- Lower Cost of Ownership:
- Easier to deploy
- Detect attacks related to sql injection and cross site scripting
- Real Time detection and quick response
- Detection of failed attacks
- This system does not do any changes in the source code of the application.
- The detection and mitigation of the Attack is fully automated.
- By increasing the number of proxy servers the web application can Handle any number of requests without obvious delay in time and still can protect the application From SQL injection attack
- Does not require additional hardware

CONCLUSIONS

This system does not do any changes in the source code of the application. The detection and mitigation of the Attack is fully automated.

By increasing the number of proxy servers the web application can Handle any number of requests without obvious delay in time and still can protect the application From SQL injection attack

REFERENCES

1. Atul Choudhary, M. L. Dhore (2012)"CIDT: Detection of malicious code injection attacks on web application", international journal of computer applications(0975-8887)
2. S. Fouzul Hidhaya, Angelina Geetha (2012)" Intrusion Protection against SQL Injection Attacks Using a Reverse Proxy", Natarajan Meghanathan, et al. (Eds): SIPM, FCST, ITCA, WSE, ACSIT, CS & IT 06, pp. 129–144, 2012. © CS & IT-CSCP 2012
3. David Litchfield, (2005) "Data-mining with SQL Injection and Inference", Next Generation Security software Ltd., White Paper.
4. Allaire Security Bulletin, (1999) "Multiple SQL statements in dynamic queries".

5. Gregory Buehrer, Bruce W. Weide and Paolo A. G. Sivilotti, (2005) “Using Parse Tree Validation to Prevent SQL Injection Attacks”, Proc. International Workshop on Software Engineering and Middleware, pp. 106-113.
6. C. Gould, Z. Su and P. Devanbu, (2004) “JDBC Checker: A Static Analysis Tool for SQL/JDBC Application”, Proc. International Conference on Software Engineering ‘04, pp.697-698.
7. W. G. Halfond and A. Orso, (2005) “AMNESIA: Analysis and Monitoring for Neutralizing SQL Injection Attacks”, Proc. ACM International Conference on Automated Software Engineering ’05, pp. 174-183.
8. Gregory Buehrer, Bruce W. Weide and Paolo A. G. Sivilotti, (2005) “Using Parse Tree Validation to prevent SQL Injection Attacks”, proc. International Workshop on Software Engineering and Middleware, pp. 106-113.
9. Zhendong Su, Gary Wassermann, (2006) “The Essence of Command Injection Attacks in Web Applications”, Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages ’06, pp.372-382.
10. Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee and S. Y. Kuo, (2004) “Securing Web Application Code by Static Analysis and Runtime Protection”, Proc. International World Wide Web Conference ’04, pp.40-52.
11. Cesar Cerrudo, (2002) “Manipulating Microsoft SQL Server Using SQL Injection”, Application Security Inc., White Paper.
12. Ofer Maor and Amichai Shulman, (2002) “Blindfolded SQL injection”, Imperva Inc., White paper.
13. W. Halfond, J. Vigeas and A. Orso, (2006) “A Classification of SQL Injection Attacks and Counter Measures”, Proc. International Symposium on Secure Software Engineering ’06